Applications of SHOP and SHOP2

Dana Nau
University of Maryland

Tsz-Chiu Au
University of Maryland

Okhtay Ilghami
University of Maryland

Ugur Kuter
University of Maryland

Héctor Muñoz-Avila
Lehigh University

J. William Murdock
IBM Research

Dan Wu
University of Maryland

Fusun Yaman
University of Maryland

CS-TR-4604, UMIACS-TR-2004-46
June 25, 2004
Revised December 12, 2004

Abstract

SHOP and SHOP2 are HTN planning systems that were designed with two goals in mind: to investigate some research issues in automated planning, and to provide some simple, practical planning tools. They are available as open-source software, and have developed an active base of users in government, industry, and universities. SHOP2 received one of the top four awards in the 2002 International Planning Competition [4].

This paper summarizes how SHOP and SHOP2 work, describes some of the applications that users have developed for them, and discusses directions for future work.

1 Introduction

The SHOP and SHOP2 are automated-planning systems were designed with two goals in mind: to investigate some research issues in automated planning, and to provide some simple, practical planning tools. They are available as open-source free software, and been downloaded thousands of times, and have been used in dozens (possibly hundreds) of projects. They have an active set of users which include government laboratories, industrial R&D projects, and academic settings. In addition, SHOP2 received one of the top four awards in the 2002 International Planning Competition [4].

SHOP and SHOP2 are based on a planning forma1lism called Hierarchical Task Network (HTN) planning. HTN planning is done by applying *HTN methods*, which basically are forms that describe how to decompose tasks into subtasks. HTN methods can be used to describe the "standard operating procedures" that one would normally use to perform tasks in some domain; thus they often correspond well to the way that users think about problems.

Unlike most other HTN planners, SHOP and SHOP2 use a search-control strategy called *ordered task decomposition*, which involves planning for the tasks in the same order that the tasks are to be accomplished in the resulting plan. Ordered task decomposition reduces the complexity of reasoning by eliminating a great deal of uncertainty about the world, making it easy to incorporate a great deal of expressive power into the planning system: for example, SHOP and SHOP2 can do complex inferential reasoning, mixed symbolic/numeric computations, and call user-supplied subroutines.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
12 DEC 2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Applications of SHOP and SHOP2	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Maryland,8400 Baltimore Avenue,College Park,MD,20742**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**14**	

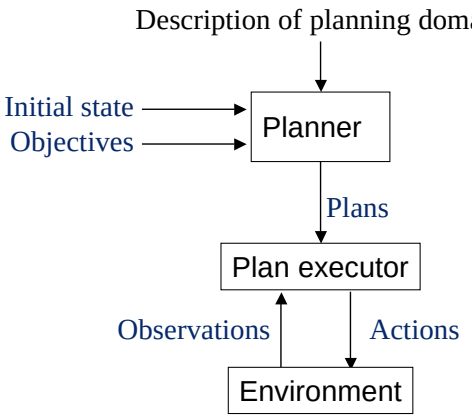


Figure 1: A simple conceptual model for planning.

This paper is organized as follows. Section 2 gives an informal description of HTN planning. Section 3 gives an overview of SHOP and SHOP2, and Section 4 summarizes some of the projects in which they have been used. Section 5 contains concluding remarks, and describes our ongoing and future work.

2 Background

Automated Planning. In general, the purpose of an automated planning system is to generate a plan or policy that a plan executor can execute in order to achieve some set of goals or objectives. Most planning research has focused on *offline* planning (see Figure 1), in which the entire plan is formulated before the plan executor begins executing it. Thus at planning time, no direct information is available to the planner about a plan's success or failure—instead, the planner must reason about whether the plan will (or is likely to) succeed or fail. For additional information, see [6].

Automated planning systems can be classified into the following categories, based on whether—and in what way—they can be configured to work in different planning domains:

- **Domain-independent planners.** In these planning systems, the sole input to the planner is a description of a planning problem to solve, and the planning engine must be general enough to work in any planning domain within some large class of planning domains $\mathcal{D}$.
  Historically, $\mathcal{D}$ was often assumed to be the set of all "classical" planning domains [6], a class that is too restricted to include most practical planning applications. However, this assumption is being used less frequently as automated-planning researchers become more interested in extending their work beyond classical planning.
- **Tunable domain-independent planners.** These are similar to domain-independent planners, but can be tuned for better performance in some planning domains. For example, LPG [5] can be tuned either (1) to run quickly without caring much about the length of the plans it finds, (2) to find short plans even if it takes a long time to find them, or (3) something in between. In the 2002 International Planning Competition [4], LPG was run with all three settings, producing three sets of results.
- **Domain-configurable planners.** These are planning systems in which the planning engine is domain-independent but the input to the planner includes domain-specific *control knowledge*, i.e.,

information to guide the planning engine in solving problems in the relevant problem domain. The planners have sometimes been called "hand-tailored" planners, but since the planning engine is domain-independent, the terms "hand-tailorable" or "control-intensive" are more accurate. Examples of such planners include HTN planners such as O-Plan [19], SIPE-2 [20], SHOP [13], and SHOP2 [12], and planners such as TLPlan [2] and TALplanner [11] in which the control knowledge consists of pruning rules.

- **Domain-specific planners.** These planning systems are tailor-made for use in a given planning domain, and are unlikely to work in other domains unless major modifications are made to the planning system. This class includes most of the planners that have been deployed in practical applications; one example is the planning algorithm for declarer play used in *Bridge Baron* [17].

HTN Planning. For domain-specific and domain-configurable planning, one of the best-known approaches is *HTN planning*, in which the planning system formulates a plan by decomposing *tasks* (symbolic representations of activities to be performed) into smaller and smaller subtasks until *primitive* tasks are reached that can be performed directly. The basic idea was developed in the mid-70s [15, 18], and the formal underpinnings were developed in the mid-90s [3].

An HTN planning problem consists of the following: the *initial state* (a symbolic representation of the state of the world at the time that the plan executor will begin executing its plan), the *initial task network* (a set of tasks to be performed, along with some constraints that must be satisfied), and a *domain description* that contains the following:

- A set of *planning operators* that describe what kinds of actions (i.e., what kinds of primitive tasks) the plan executor can perform. Each operator may have a set of preconditions that must be true in the state in which the operator is to be executed, and a set of effects that will occur when the operator is executed. An action (or synonymously, a primitive task) is an operator instance, produced by assigning values to an operator's parameters.
- A set of *methods* that describe various possible ways of decomposing tasks into subtasks. These are the "standard operating procedures" that one would normally use to perform tasks in the domain. Each method may have a set of constraints that must be satisfied in order to be applicable.
- Optionally, various other information such as definitions of auxiliary functions and definitions of axioms for inferring conditions that are not mentioned explicitly in states of the world.

Planning is done as follows. For each nonprimitive task, the planner chooses an applicable method and instantiates it to decompose the task into subtasks. For each primitive task, the planner chooses an applicable operator and instantiates it to produce an action. If all of the constraints are satisfied, then the planner has found a solution plan; otherwise the planning system will need to backtrack and try other methods or other instantiations.

Example. Figure 2 gives a pseudocode representation of an extremely simple planning domain in which there are two ways to travel from one location to another: by foot and by taxi. These are represented by two methods: travel-by-foot and travel-by-taxi. The travel-by-foot method has one constraint: a precondition saying that the distance from the starting point to the destination must be less than or equal to 2 miles. If the method is applicable, it decomposes the task into a single subtask: walk to the park. The travel-by-taxi method has one constraint, which is also a precondition: the traveler must have enough cash to pay the taxi driver. If the method is applicable,

method travel-by-foot
  precond: $distance(x, y) \leq 2$
  task: travel(a, x, y)
  subtasks: walk(a, x, y)

method travel-by-taxi
  task: travel(a, x, y)
  precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
  subtasks: call-taxi$(a, x) \longrightarrow$ ride$(a, x, y) \longrightarrow$ pay-driver(a, x, y)

operator walk
  precond: $location(a) = x$
  effects: $location(a) \leftarrow y$

operator call-taxi(a, x)
  effects: $location(taxi) \leftarrow x$

operator ride-taxi(a, x)
  precond: $location(taxi) = x$, $location(a) = x$
  effects: $location(taxi) \leftarrow y$, $location(a) \leftarrow y$

operator pay-driver(a, x, y)
  precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
  effects: $cash(a) \leftarrow cash(a) - 1.5 + 0.5 \times distance(x, y)$

Figure 2: Pseudocode representation of a simple travel-planning domain. Left-arrows denote assignments of values to state-variables; right-arrows are ordering constraints.

it decomposes the task into three subtasks: call a taxi, ride to the park, and pay the driver. All of the subtasks are primitive, i.e., the traveler is expected to know how to accomplish them directly.

Now, suppose that in the initial state, I am at home, I have $20, and I want to travel to a park that is 8 miles away. To plan how to travel to the park (see Figure 3), first I try to use the travel-by-foot method, but this method is not applicable because the park is more than 2 miles away. Next, I try to use the travel-by-taxi method. Its precondition is satisfied, so the method produces a sequence of three subtasks, with a constraint saying they are to be performed in the following order: (1) call a taxi to my home, (2) ride in it to the park, and (3) pay the driver $5.50. The subtasks all are primitive, i.e., each of them corresponds to an action. The first action has no preconditions, so it is applicable and produces a state s_1 that is identical to the initial state except that $location(taxi) = home$. This state satisfies the preconditions of the second action. The second action produces a state in which the precondition of the third action is satisfied, so I have a solution plan. After execution of this plan, the final state will be as shown in the figure.

3 SHOP and SHOP2

HTN planning is basically a trial-and-error search: the planner may have to try many different possibilities before finding a plan that works. In any trial-and-error search, one of the most important questions is what kind of search-control strategy to use.

SHOP2 uses a search-control strategy called *ordered task decomposition*. It decomposes tasks into subtasks in the same order that the tasks are to be accomplished. For example, to produce the decomposition tree in Figure 4(a), SHOP2 would decompose the tasks in the order

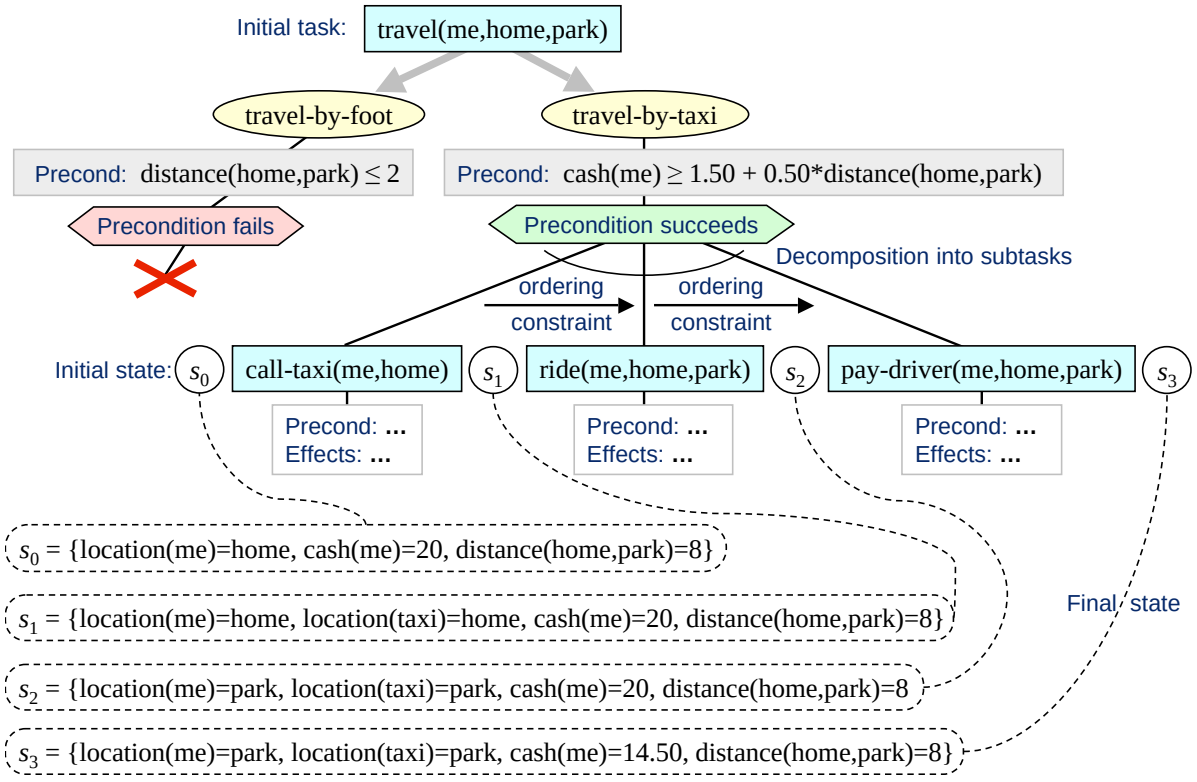
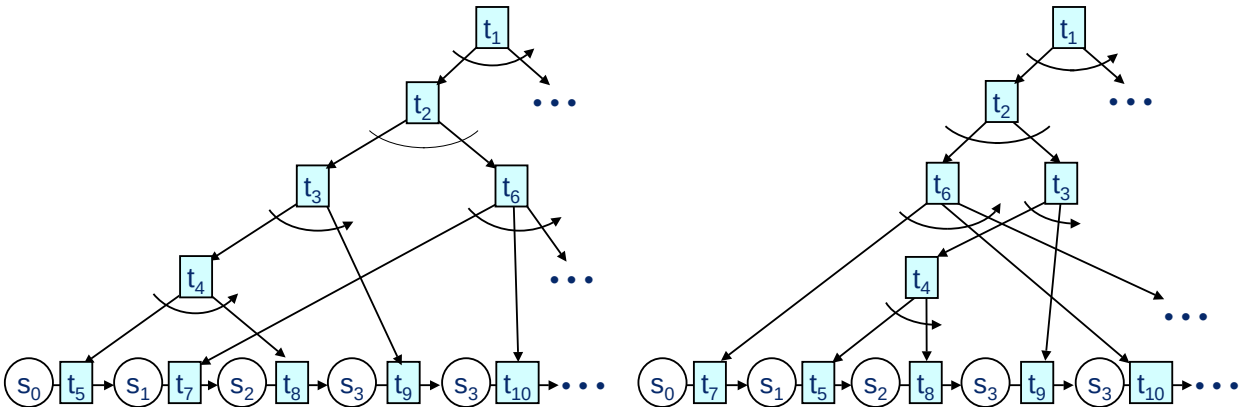


Figure 3: Solving a planning problem in the travel-planning domain.



(a) (b)

Figure 4: Two examples of task decomposition with interleaved subtasks. Arrows on the arcs indicate ordering constraints. There is no ordering constraint between t_3 and t_6, hence they may be performed in either order and their subtasks may be interleaved (provided, of course, that each task's preconditions are satisfied).

$t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}$, and to produce the decomposition tree in Figure 4, SHOP2 would decompose the tasks in the order $t_1, t_2, t_6, t_7, t_3, t_4, t_5, t_8, t_9, t_{10}$.

SHOP, the predecessor of SHOP2, is like SHOP2 except that it requires a strict linear ordering on subtasks, hence subtasks cannot be interleaved. For example, SHOP could not produce the decomposition trees in Figure 4 because it would not be able to interleave t_3's and t_6's subtasks. Thus, some planning domains can be described much more easily in SHOP2 than in SHOP.

Ordered task decomposition always generates the actions of a plan in the same order that the plan executor will execute those actions. Thus, the current state is known at each step of the planning process. Since it is easier to reason about what *is* true than what *might* be true, this has made it easy to incorporate complex reasoning capabilities into SHOP and SHOP2, such as the auxiliary functions and axioms mentioned earlier.

In April 2002, SHOP2 achieved high visibility because of its performance in the 2002 International Planning Competition, where it received one of the top four awards.[1] SHOP2 was one of the three fastest planners in the competition. Its reasoning capabilities enabled it to generate much smaller search spaces than those of most of the other systems. Hence it could solve planning problems many times more quickly, and could solve much more complicated problems. SHOP2 solved 899 out of 904 problems, more than any of the other systems.

SHOP and SHOP2 are open-source software. Common Lisp and Java implementations are available at `http://www.cs.umd.edu/projects/shop`.

4 Applications of SHOP and SHOP2

SHOP and SHOP2 have been downloaded several thousand times,[2] and have a significant user base that includes government laboratories, industries, and universities. Based on information provided by some of those users, here are descriptions of some of the projects in which SHOP and SHOP2 have been used.

4.1 Projects in Government Laboratories

Evacuation Planning. The HICAP system at the US Naval Research Laboratory (NRL) is a tool for helping experienced human planners to develop evacuation plans, i.e., plans to evacuate humans whose lives are in danger. Evacuation planning must be done by a human expert or under the supervision of a human expert: it is unrealistic to expect that a planning system could produce good plans by itself, and flawed evacuation plans could yield dire consequences. For this reason, the top level of HICAP is a plan editor with which users can edit tasks manually and can use a planning system to interactively refine plans.

Only part of the knowledge necessary for evacuation planning can be formalized sufficiently for automated planning. In general, there will be an incomplete domain description, in the form of standard requirements and operating procedures—but these are not sufficient for deriving detailed plans. For that, knowledge about previous experiences is essential. Thus, HICAP's planning module includes both generative and case-based planning. The generative component is provided

[1] There were two awards for "distinguished performance" and two for "distinguished performance of the first order;" SHOP2 received one of the former [4].

[2] As of December 11, 2004, our web site's log shows 2163 downloads, but this does not include the times when users downloaded directly from our ftp server rather than going through our web interface. We imagine the total number of downloads exceeds 2500.

by SHOP. The case-based component, NaCoDAE, works by retrieving fragments of plans from previous evacuations. Within HICAP, NaCoDAE and SHOP are integrated quite tightly: each is capable of using the other to decompose tasks into subtasks.

Contact: David Aha, Naval Research Laboratory, Washington, DC.

URL: `http://www.aic.nrl.navy.mil:80/hicap`

Evaluating Terrorist Threats. The purpose of the Naval Research Laboratory's AHEAD project is to help intelligence analysts understand and evaluate hypotheses about terrorist threats. Given a hypothesis, the AHEAD system uses analogical retrieval to obtain a model of the hostile activity most closely related to the hypothesis. This model is encoded as an HTN domain description for SHOP2 in which individual actions are annotated with additional explanatory information about their function. AHEAD invokes SHOP2 using this domain description to produce a plan that is compatible with the hypothesis. As each operator is added to the plan, SHOP2 queries an external evidence database to determine whether the evidence is consistent with that operator. When the evidence is consistent with the operator, AHEAD generates an argument in favor of the hypothesis; when the evidence is inconsistent, AHEAD generates a counterargument. The resulting structured argument is presented in a browsable user interface. HTN planning is particularly well-suited to this process because HTNs organize behavior into meaningful components at multiple levels of abstraction thus enabling coherent, structured argumentation.

Contact: David Aha, Naval Research Laboratory, Washington, DC.

URL: `http://www.nrl.navy.mil/aic/iss/ida/projects/ahead/AHEAD.php`

Fighting Forest Fires. The European Union's COMETS project focuses on the development of unmanned aerial vehicle (UAV) control techniques for detection and monitoring of forest fires. As part of this project, researchers at LAAS, a government research laboratory in Toulouse, France, are developing a distributed architecture in which each UAV will contain a generic "decisional node" consisting of a supervisor and a planner.

Within each decisional node, they are using SHOP2 as the symbolic planner: they exploit SHOP2's forward-chaining capability to integrate its planning activity with specialized software for estimating the costs, time, etc., for basic UAV operations. In order to perform temporal reasoning, they are using the same time-stamping technique we developed for temporal planning with SHOP2 in the 2002 International Planning Competition [12]. The researchers anticipate that they soon will have simulation results and will be able to run experiments using LAAS's blimp, *Karma*.

Contact: Jeremi Gancet, Simon Lacroix, and Raja Chatila, LAAS/CNRS, Toulouse, France.

URL: `http://www.comets-uavs.org`

Software Systems Integration. NIST (National Institute of Standards and Technology) is using SHOP2 in a project whose goal is to automate tasks of software systems integration. So far, the particular example they have used is based on General Motors' ebXML-based "bulk rental car buying" interfaces. These interfaces allow a buyer to search for cars of a particular make, model and year, and purchase them. But if a buyer wants to get a summary of various cars at various locations (e.g., in order to minimize costs), the interface makes it difficult to do this. NIST's code reads the seller's ebXML BPSS (business process specification schema) and produces a planning problem in which SHOP determines the sequence of transactions against the seller's interfaces to achieve the buyer's objective.

Contact: Peter Denno, NIST, Gaithersburg, MD.

URL: `http://www.mel.nist.gov/proj/mee.htm`

4.2 Industry Projects

Controlling Multiple UAVs. SIFT, LLC is using a modified version of the SHOP2 planner in a UAV control system in their PVACS project, with funding from DARPA through an SBIR contract. SIFT's "Playbook" control system allows time-pressured users, who are not UAV operators, to request reconnaissance missions using high-level tasking commands, modeled on the way people delegate tasks to human subordinates. The SIFT Playbook supports interactions through both PDA and desktop/laptop interfaces. The Playbook translates users' brief, general commands into very specific control actions suitable for execution. The Playbook's Executive provides high-level closed-loop monitoring and implementation of the Playbook's plans, controlling multiple UAVs through the Variable Autonomy Control System (VACS) Ground Control Station (GCS), developed by Geneva Aerospace, Inc. The Playbook currently operates these UAVs in a high-fidelity simulation environment, but the interface it uses to control the simulated UAVs through the VACS GCS is the same as the one used to direct VACS UAVs in real flight operations.

The modified SHOP2 planner plays a key role in SIFT's Playbook, translating the user's high level task specifications into a sequence of commands that can be executed by UAVs. SIFT's plan library contains tasks for multiple reconnaissance missions, for both rotorcraft and fixed-wing UAVs. Robert Goldman at SIFT has developed an augmented version of SHOP2 that generates temporal plans including durative actions, and provides more knowledge-engineering and debugging support.

Contact: Robert Goldman, SIFT, Minneapolis, MN.

URL: `http://www.sift.info/English/projects/PVACS.htm`

Evaluation of Enemy Threats. Lockheed Martin Advanced Technology Laboratories, in collaboration with the Army Research Laboratory, is using SHOP in a project that attempts to evaluate possible enemy threats. They are using SHOP to decompose higher level tasks such as 'attack blue-convoy' into sequences of operations such as 'move red-tank1 to location2, ..., fire red-tank1 at blue-convoy.' Due to their confidentiality restrictions, they could not give us further details.

Contact: Benjamin Grooters and Sergio Gigli, Lockheed Martin ATL, Cherry Hill, NJ.

Location-Based Services. Sony Electronics Incorporated has used SHOP in a project aimed at developing mobile GIS devices to help people plan errands that take them to different geographical locations. Due to Sony's confidentiality requirements, they could not give us further details.

Contact: Mark Plutowski, Sony Electronics Incorporated, San Jose, CA.

Material Selection for Manufacturing. Infocraft Ltd. is developing a system that uses SHOP2 for material selection in continuous-process manufacturing: specifically, the production of activated carbon from charcoal using a discrete set of continuous manufacturing processes. The desired properties of the carbon (specifically its grade size and adsorption level) will vary from one run to another, as will the characteristics of different supplies of charcoal. The objective of the project is to use SHOP2 to select which supplies of charcoal will most reliably produce activated carbon with

a desired set of properties. SHOP2's abilities to do numerical and axiomatic reasoning are essential for this project: adsorption levels are represented as real numbers, and grade sizes are represented as normal distributions.

Contact: Nuwan Waidyanatha, Infocraft Ltd., Sri Lanka.

URL: `http://www.infocraft.lk`

4.3 University Projects

Automated Composition of Web Services. Web services are Web accessible, loosely coupled chunks of functionality with an interface described in a machine readable format. Web services are designed to be *composed*, that is, combined in workflows of varying complexity to provide functionality that none of the component services could provide alone.

In the OWL-S (formerly DAML-S) language for semantic markup of web services, services can be described as complex or atomic processes with preconditions and effects. This makes it possible to translate the OWL-S process-model constructs directly to SHOP2 methods and operators, and we have developed an algorithm to do so. This means that SHOP2 can be used to solve service composition problems, by telling SHOP2 to find a plan for the task that is the translation of a composite process [21, 16].

Contact: Evren Sirin and James Hendler, Dept. of Computer Science, University of Maryland.

URL: `http://www.mindswap.org/~evren/composer`

Project Planning. The SHOP/CCBR system is a tool developed at the Lehigh University for investigating the use of HTN planning techniques to support project management. The SHOP/CCBR system is a straightforward extension of SHOP that uses *cases* to decompose tasks. Cases are similar in structure to methods, the main difference being that cases include preference information for use in ranking applicable cases. SHOP/CCBR uses a communication module to interact with Microsoft Project, a commercial tool for project management. This allows displaying the HTN decompositions generated with SHOP's hierarchical planning algorithm in Microsoft Project. The on-going work involves developing algorithms to capture cases automatically from user interactions with Microsoft Project.

Contact: Héctor Muñoz-Avila, Computer Science and Engineering, Lehigh University.

URL: `http://www.cse.lehigh.edu/~munoz/projects/KBPP`

Statistical Goal Recognition in Agent Systems. For an agent to perform effectively in a multi-agent environment, an important task is *goal recognition*, i.e., inferring the goals of other agents. Researchers at the University of Rochester are developing a statistical approach to goal recognition using machine-learning techniques. To do the learning requires a labeled "plan corpus" of plans and their associated goals. They are using SHOP2 to generate such plan corpora stochastically. For this purpose, they are using a modified version of SHOP2 that makes random choices at every point where more than one possible decision is available to SHOP2.

Contact: Nate Blaylock and James Allen, Computer Science Dept., University of Rochester.

URL: `http://www.cs.rochester.edu/research/cisd/projects/goalrec`

Distributed Planning. Researchers at the University of Sherbrooke are developing a general approach for turning backtracking-search-based planners into distributed planning systems that run on networks of clusters. The basic idea is to distribute backtrack search points to different processes on the network. To implement their approach, they have developed DSHOP, a distributed version of SHOP.

Contact: Froduald Kabanza, Département dinformatique Université de Sherbrooke, Canada.

URL: `http://www.dmi.usherb.ca/~kabanza/sharcnet`

Additional University Projects. Worldwide, SHOP and SHOP2 have been used in many more college and university projects than we can mention, but here are some notes about a few of them.

- Drexel University regularly uses SHOP and SHOP2 in their Introductory AI class in order to teach planning, and in their Knowledge-Based Agents course to do agent reasoning, service composition, etc. *Contact:* William Regli, `regli@drexel.edu`.
- At the National University of Colombia in Medellin, Colombia, a system is being developed that uses SHOP2 to automatically create virtual courses from existing educational material. *Contact:* Néstor Daro Duque Mendez, `nduque@nevado.manizales.unal.edu.co`.
- At the Technical University of Cluj-Napoca in Romania, SHOP has been used for an e-commerce application, to build plans for bidding in a modified version of the Trading Agent Competition. *Contact:* Adrian Groza, `adrian.groza@email.ro`.
- At Trinity College Dublin, SHOP2 is being used in a web-service composition project somewhat similar to ours. *Contact:* Dónal Murtag, `domurtag@cs.tcd.ie`.
- In the University of Maryland's Aerospace Engineering Department, SHOP2 is being used as the planning component in an architecture that combines task planning, real-time scheduling, and motion/trajectory planning. *Contact:* Ella Atkins, `atkins@glue.umd.edu`.
- At Villanova University, SHOP2 has been used in a mock spacecraft-mission scenario, to study how the density, distribution and overall layout of environment obstacles can be used to compute and predict the best optimization technique to use within SHOP2. *Contact:* Filip Jagodzinski, `filip.jagodzinski@villanova.edu`.
- At University of Arizona, SHOP is being used to generate process alternatives within a case-based reasoning framework for business workflow management. *Contact:* Madhu Therani, `madhu@email.arizona.edu`.

5 Concluding Remarks

We have been pleasantly surprised at the extent to which people have begun using SHOP and SHOP2 in their research and development projects. We believe that this has come about for several reasons:

- SHOP and SHOP2 are based on HTN decomposition. This seems to correspond well to the way in which their users think about how to generate plans.
- Unlike most other automated-planning systems, SHOP and SHOP2 plan for tasks in the same order that the tasks will be executed. This has made it easier to understand what the planners are doing, and to incorporate complex reasoning into the domain knowledge.
- SHOP and SHOP2 are available as open-source software. This has made it easy for users to find and fix bugs, and to adapt the software for their own purposes.

We have many ideas for extensions and improvements, and we now describe some of them.

5.1 Automated Learning of Planning Domains

A great challenge in using any planning system to solve real-world problems is how to acquire the domain knowledge that the system will need. We can divide this domain knowledge into two types:

- *The domain definition,* i.e., what the states and operators are. Researchers usually assume that such knowledge will be provided *a priori*, but in some real-world domains we are likely to have only a partial or approximate description of the planning domain. For example, it is easy to say that a drilling operation will drill a hole, but it is much harder to say how far the hole will deviate from perfect straightness and roundness, or to model the conditions under which the drill bit is likely to break [14].
- *Control knowledge*, i.e., information to guide the planning process (see Section 2). In HTN planners such as SHOP and SHOP2, this knowledge consists of methods. In planning systems such as TLPlan [2] and TALplanner [11], it consists of rules for pruning the search space.

We are working on ways to acquire HTN methods automatically by having the planning system *learn* them from plan traces. We have developed a general formal framework for learning HTN methods, and a supervised learning algorithm named CaMeL that is based on this formalism. In [8] we present theoretical results about CaMeL's soundness, completeness, and convergence properties, and experimental studies of its speed of convergence under different conditions. The experimental results suggest that CaMeL may potentially be useful in real-world applications.

5.2 Compiling Planning Domains

A domain-configurable planner may be viewed as an interpreter of its domain-description language: given a domain description D and a planning problem P, the planner invokes the methods and operators of D interpretively on P. An alternative approach is to write a *compiler* for the domain description language: the input to the compiler is a domain description D, and the output is a domain-specific planning program for D, that can be run directly on any planning problem P in D.

The advantages of such a *planner compilation* approach are analogous to the advantages that compilation has over interpretation in conventional programming languages. By compiling domain descriptions directly into low-level executable code, we can do implementation-level optimizations that are not otherwise possible and have not been explored in previous research on AI planning. These optimizations can be coupled with other speed-up techniques (e.g., domain analysis and other automated domain information synthesis techniques) in order to obtain additional speedups.

We are developing JSHOP2, a Java implementation of SHOP2 that uses this domain-compilation technique. Our preliminary experimental results suggests that the compilation technique increases the planner's efficiency by an order of magnitude [7]. JSHOP2 is available at `http://www.cs.umd.edu/projects/shop`.

5.3 Planning Under Uncertainty

Automated-planning research has traditionally assumed that each action is deterministic, i.e., it has only one possible outcome. However, in many situations it is more realistic to allow actions to

be *nondeterministic*, i.e., to have more than one possible outcome. For example, the outcome of an action might vary because of the responses of other agents, or because of random changes in the environment.

We have developed a general technique for taking planners such as SHOP, SHOP2, TLPlan, and TALplanner, and adapting them to deal with nondeterministic actions. We have shown both theoretically and experimentally [9] that our approach can produce exponential speedups over previous algorithms for planning in nondeterministic environments.

We are currently extending our approach to work in Markov decision processes (MDPs), in which the actions have probabilistic outcomes. Our preliminary results show that we can obtain similar speedups here as well.

5.4 Planning with Distributed Information Sources

Planning researchers typically assume that the planning system is *isolated*: it begins with a complete description of the planning problem, and has no need of interacting with the external world during the planning process. In many practical situations, such an assumption is clearly unrealistic: the planner may need to obtain information from external sources during planning.

We have developed a formalism for *wrappers* that may be placed around conventional (isolated) planners to replace some of the planner's memory accesses with queries to external information sources. When appropriate, the wrapper can automatically backtrack the planner to a previous point in its operation. We have done both mathematical and experimental analysis of several different query-management strategies for these wrappers, i.e., strategies for when to issue queries, and when/how to backtrack the planner [1]. We show conditions under which different query management strategies are preferable, and suggest that domain-configurable planners such as SHOP2 are likely to be better suited than other planners for planning with volatile information.

Even better performance can be obtained if a planner can make *non-blocking* queries to external information sources, i.e., if the planner can continue exploring other parts of its search space while waiting for the response to a query. We have developed a modified version of SHOP2 that works in this way, and have shown experimentally that this dramatically improves the planner's performance [10].

5.5 Temporal Planning

One weakness of SHOP2 is that its methods and operators do not explicitly represent time and concurrency—two things that would be essential, for example, to have the actions of different agents (e.g. multiple subordinates or team members) going on in parallel. In some cases, it is possible to overcome this limitation by writing planning operators that explicitly assert time-stamp information into the current state of the world [12]. On the other hand, it clearly would be advantageous to have a more comprehensive way to represent and reason about time and concurrency. We hope to address this problem in our future work.

Acknowledgments

We are grateful to our community of users for the information they gave us about how they are using SHOP and SHOP2 in their projects.

This work was supported in part by the following grants and contracts: Air Force Research Laboratory F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, Naval Research Laboratory N00173021G005, and National Science Foundation IIS0412812. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

[1] T.-C. Au, D. Nau, and V. Subrahmanian. Utilizing volatile external information during planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 647–651, August 2004.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[3] K. Erol, J. Hendler, and D. S. Nau. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93, 1996.

[4] M. Fox and D. Long. International planning competition, 2002. `http://planning.cis.strath.ac.uk/competition`.

[5] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, pages 968–973, 2002.

[6] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, May 2004.

[7] O. Ilghami and D. S. Nau. A general approach to synthesize problem-specific planners. Technical Report CS-TR-4597, UMIACS-TR-2004-40, University of Maryland, October 2003.

[8] O. Ilghami, D. S. Nau, H. Muñoz-Avila, and D. W. Aha. CaMeL: Learning methods for HTN planning. In *AIPS-2002*, Toulouse, France, 2002.

[9] U. Kuter and D. Nau. Forward-chaining planning in nondeterministic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 513–518, July 2004.

[10] U. Kuter, E. Sirin, D. Nau, B. Parsia, and J. Hendler. Information gathering during planning for web services composition. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *3rd International Semantic Web Conference (ISWC2004)*, volume 3298, pages 335–349. Springer-Verlag, 2004.

[11] J. Kvarnström and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Articial Intelligence*, 30:119–169, 2001.

[12] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, December 2003.

[13] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In T. Dean, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 968–973. Morgan Kaufmann Publishers, July 31–August 6 1999.

[14] D. S. Nau, W. C. Regli, and S. K. Gupta. AI planning versus manufacturing-operation planning: A case study. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[15] E. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 206–214, 1975.

[16] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 2004. To appear.

[17] S. J. J. Smith, D. S. Nau, and T. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

[18] A. Tate. Generating project networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 888–893, 1977.

[19] A. Tate, B. Drabble, and R. Kirby. *O-Plan2: An Architecture for Command, Planning and Control*. Morgan-Kaufmann, 1994.

[20] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.

[21] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, November 2003.